**Лабоpатоpная pабота №1**

**Линейные списки**

**11 Вариант**

Выполнить задания, используя для пpедставления очеpедей и стеков:

а) массивы; б) динамические списки.

Даны две очереди целых чисел от 0 до 1000. Из элементов пеpвой очереди,

котоpых нет во втоpой сфоpмиpовать стек.

Стек - структура данных, которая работает по принципу последним пришёл, первым ушёл(LIFO — last in, first out).

Очередь - структура данных, которая работает по принципу первым пришёл, первым ушёл(FIFO — first in, first out).

Алгоритм:

Инициализация двух очередей que1 и que2 заданной длины.

Структура Queue1:

head: индекс головы очереди (первый элемент)

tail: индекс хвоста очереди (последний элемент)

size: размер очереди

data: массив для хранения элементов очереди

initQueue(Queue1& q, int capacity): инициализация очереди с указанной вместимостью

nullQueue(Queue1& q): сброс очереди

next(Queue1& q, int n): вычисление следующего индекса в очереди

empty(Queue1& q): проверка, пуста ли очередь

add(Queue1& q, int value): добавление элемента в очередь

del(Queue1& q): извлечение элемента из очереди

printQueue(Queue1& q): вывод содержимого очереди

Заполнение первой очереди que1 последовательными числами от 0 до quel1-1.

Заполнение второй очереди que2 числами со значением 1.

Вывод содержимого обеих очередей на экран.

Создание пустого стека stack.

Она содержит следующие поля:

top: индекс верхнего элемента стека

data: массив для хранения элементов стека

capacity: максимальный размер стека

initStack(Stack1& st, int capacity): инициализация стека с указанной вместимостью

push(Stack1& st, int value): добавление элемента в стек

pop(Stack1& st): извлечение элемента из стека

nullStack(Stack1& st): сброс стека

empty(Stack1& st): проверка, пуст ли стек

printStack(Stack1& st): вывод содержимого стека

Инициализация и заполнение двух очередей.

Создание пустого стека.

Последовательная обработка элементов первой очереди:

Извлечение элемента из первой очереди.

Проверка наличия этого элемента во второй очереди.

Если элемент отсутствует во второй очереди, добавление его в стек.

Вывод содержимого стека.

**а)**

#include <iostream>

#include <time.h>

using namespace std;

struct Stack1 {

int top;

int\* data;

int capacity;

};

void initStack(Stack1& st, int capacity) {

st.data = new int[capacity];

st.top = -1;

st.capacity = capacity;

}

void push(Stack1& st, int value) {

if (st.top + 1 < st.capacity) {

st.data[++st.top] = value;

}

}

int pop(Stack1& st) {

return st.data[st.top--];

}

void nullStack(Stack1& st) {

st.top = -1;

}

bool empty(Stack1& st) {

return st.top == -1;

}

void printStack(Stack1& st) {

for (int i = st.top; i >= 0; i--) {

cout << st.data[i] << " ";

}

}

struct Queue1 {

int head, tail, size;

int\* data;

};

void nullQueue(Queue1& q) {

q.head = 0; q.tail = q.size - 1;

}

void initQueue(Queue1& q, int capacity) {

q.size = capacity + 1;

q.data = new int[q.size];

nullQueue(q);

}

int next(Queue1& q, int n) {

return (n + 1) % q.size;

}

bool empty(Queue1& q) {

return next(q, q.tail) == q.head;

}

void add(Queue1& q, int value) {

if (next(q, next(q, q.tail)) == q.head) {

cout << "Queue overflow" << endl;

}

else {

q.tail = next(q, q.tail);

q.data[q.tail] = value;

}

}

int del(Queue1& q) {

if (empty(q)) {

cout << "Queue is empty" << endl;

return 0;

}

else {

int d = q.data[q.head];

q.head = next(q, q.head);

return d;

}

}

void printQueue(Queue1& q) {

int i = q.head;

while (i != q.tail) {

cout << q.data[i] << " ";

i = next(q, i);

}

cout << q.data[i] << " ";

}

int main() {

srand(time(0));

int quel1, quel2, x;

quel1 = 5;

quel2 = 5;

cout << "Queue length of 1 queue=" << quel1 << endl;

cout << "Queue length of 2 queue=" << quel2 << endl;

Queue1 que1, que2;

initQueue(que1, quel1);

initQueue(que2, quel2);

for (int i = 0; i < quel1; i++) {

add(que1, i);

}

for (int i = 0; i < quel2; i++) {

add(que2, 1);

}

//вывод1

cout << "queue1:" << endl;

printQueue(que1);

cout << endl;

//вывод2

cout << "queue2:" << endl;

printQueue(que2);

cout << endl;

//

bool f = 1;

int a;

Stack1 stack;

initStack(stack, que1.size);

while (!empty(que1)) {

f = 1;

a = del(que1);

int i = que2.head;

**while (i != que2.tail) {**

**if (a == que2.data[i]) f = 0;**

**i = next(que2, i);**

**}**

if (f) push(stack, a);

}

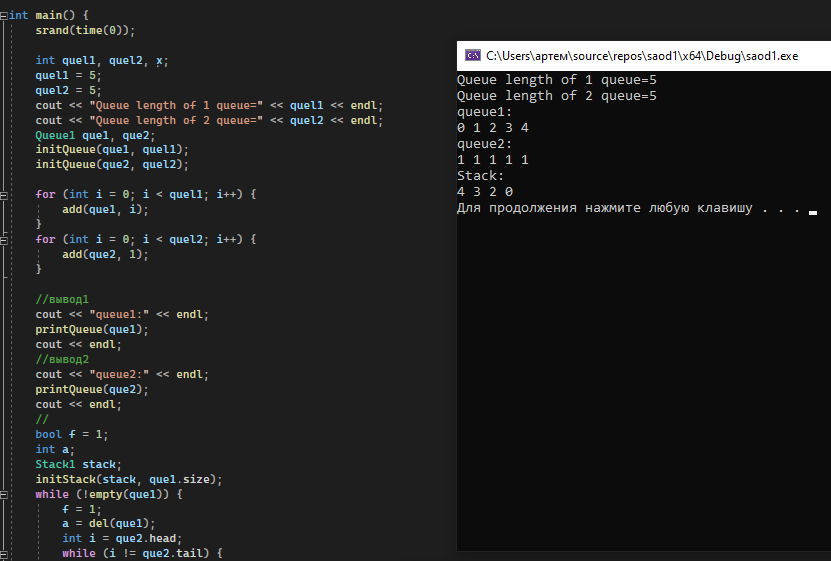
cout << "Stack:" << endl;

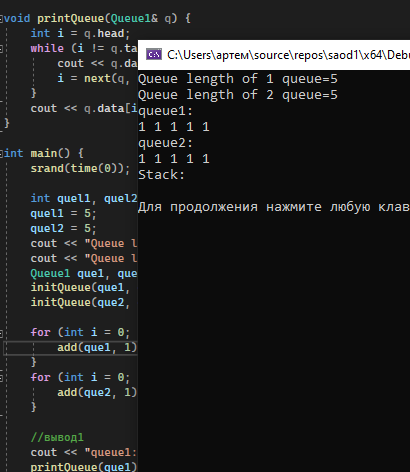
printStack(stack);

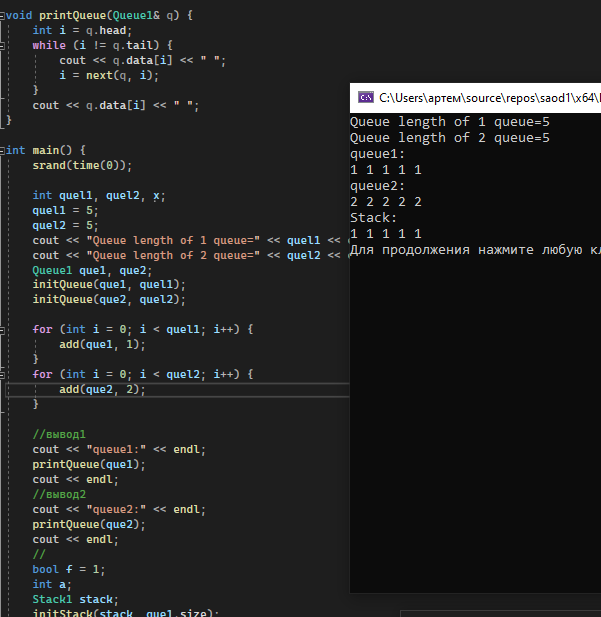
cout << endl;

system("Pause");

}

****

****

****

**б)**

Алгоритм:

Создание очередей que1 и que2

Очередь – класс Queue, который обрабатывает структуру Node как очередь структура Node имеет поля data, next, head, tail, top.

data – целочисленная переменная, содержащая значение элемента узла (Node). Узел – элемент очереди, связанный с другими элементами очереди при помощи полей next, head, tail и top. head – указатель на первый узел в очереди (изменяется при удалении элементов в очереди). next – укзатель на следующий узел в очереди. tail – указатель на последний узел очереди. top – указатель на первый элемент очереди ( не изменяется при изменении очереди)

Инициализация очередей que1 и que2

Заполнение очередей рандомным количеством рандомных элементов методом add: Если очередь пуста, то значению head присваивается адрес нового узла.

Полю data присваивается значение нового элемента.

tail = head

top = head

иначе:

значению поля next последнего узла очереди присваивается адрес нового узла.

значению tail присваивается значение адреса нового узла

Вывод очередей на экран методом ShowQueue:циклом:

если очередь не пустая, значение поля data узла, на который указывает head выводится на экран, переменной head присваивается адрес следующего после него узла.

Значению head присваивается значение top.

Объявление переменной целочисленной a и булевой переменной f

f = 1;

Стек реализован при помощи структуры Node, полями которой являются data и next. data – целочисленная переменная со значением элемента стека. next – указатель на следующий элемент стека.

Создание указателя на узел stacktop и stackstart(stacktop – указатель на верхний элемент стека, изменяющийся при обработке стека. stackstart - указатель на верхний элемент стека, не изменяющийся при обработке стека.

инициализация стека методом InitStack

Пока 1 очередь не пуста

переменной a присваивается значение поля data узла, на который указывает head методом del:

eсли очередь не пустая, значение поля data узла, на который указывает head выводится на экран, переменной head присваивается адрес следующего после него узла.

f=1

Пока 2 очередь не пуста: если a равна хотя бы одному элементу 2очереди (del), то

f=0.

head 2 очереди = top 2 очереди

Если f =1, то значение элемента a добавляется в стек методом push:

указателя на новый узел tmp

поле next узла tmp = top

top = tmp

полю data узла, на который указывает next, присваивается значение нового элемента стека

циклом:

Вывод стека на экран методом ShowStack:

Если стек не пуст, то вывод на экран верхнего элемента стека

stacktop = stackstart

empty(): проверка, пуста ли очередь

add(int value): добавление элемента в конец очереди

ShowQueue(): извлечение элемента из начала очереди

del(): удаление элемента из начала очереди

headtop(): перемещение указателя на начало очереди в top

nullQueue(): очистка очереди

InitStack(Node\*& top): инициализация пустого стека

push(Node\*& top, int value): добавление элемента в стек

ShowStack(Node\*& top): извлечение элемента из стека

empty(Node\*& top): проверка, пуст ли стек

nullStack(Node\*& top): очистка стека

Код:

#include<iostream>

#include <time.h>

using namespace std;

struct Node {

int data;

Node\* next;

};

void InitStack(Node\*& top) {

top = NULL;

}

void push(Node\*& top, int value) {

Node\* tmp = new Node;

tmp->next = top;

top = tmp;

top->data = value;

}

int del(Node\*& top) {

Node\* tmp = top;

int d = top->data;

top = top->next;

return d;

}

bool empty(Node\*& top) {

return top == NULL;

}

void nullStack(Node\*& top) {

Node\* tmp;

while (!empty(top)) {

tmp = top;

top = top->next;

delete(tmp);

}

}

class Queue {

private:

struct Node {

int data;

Node\* next;

};

Node\* head, \* tail, \* top;

public:

Queue() {

head = NULL;

tail = NULL;

}

bool empty() {

return head == NULL;

}

void add(int value) {

if (empty()) {

head = new Node;

head->data = value;

head->next = NULL;

tail = head;

top = head;

}

else {

tail->next = new Node;

tail = tail->next;

tail->data = value;

tail->next = NULL;

}

}

void headtop() {

head = top;

}

void nullQueue() {

Node\* tmp;

while (!empty()) {

tmp = head;

head = head->next;

delete(tmp);

}

}

int del() {

if (!this->empty()) {

int d = head->data;

head = head->next;

return d;

}

}

};

int main() {

srand(time(0));

Queue que1, que2;

for (int i = 0; i < rand() % 100; i++) {

que1.add(0);

}

for (int i = 0; i < rand() % 100; i++) {

que2.add(1);

}

cout << "Queue1:";

while (!que1.empty()) cout << que1.del() << " ";

que1.headtop();

cout << endl;

cout << "Queue2:";

while (!que2.empty()) cout << que2.del() << " ";

que2.headtop();

cout << endl;

int a;

bool f = 1;

Node\* stacktop, \* stackstart;

InitStack(stacktop);

while (!que1.empty()) {

f = 1;

a = que1.del();

while (!que2.empty()) {

if (a == que2.del()) f = 0;

}

if (f) {

push(stacktop, a);

}

que2.headtop();

}

stackstart = stacktop;

cout << "Stack:";

while (!empty(stacktop)) {

cout << del(stacktop) << " ";

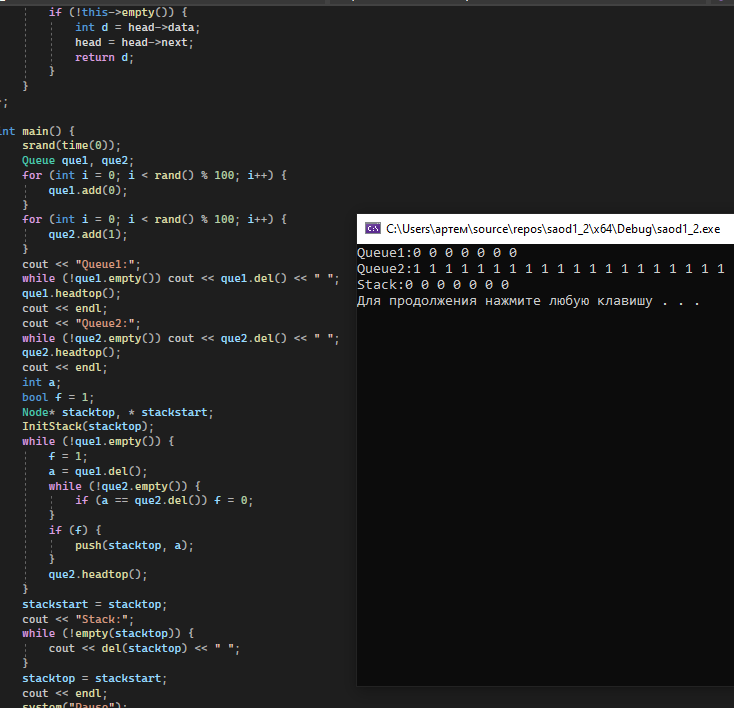
}

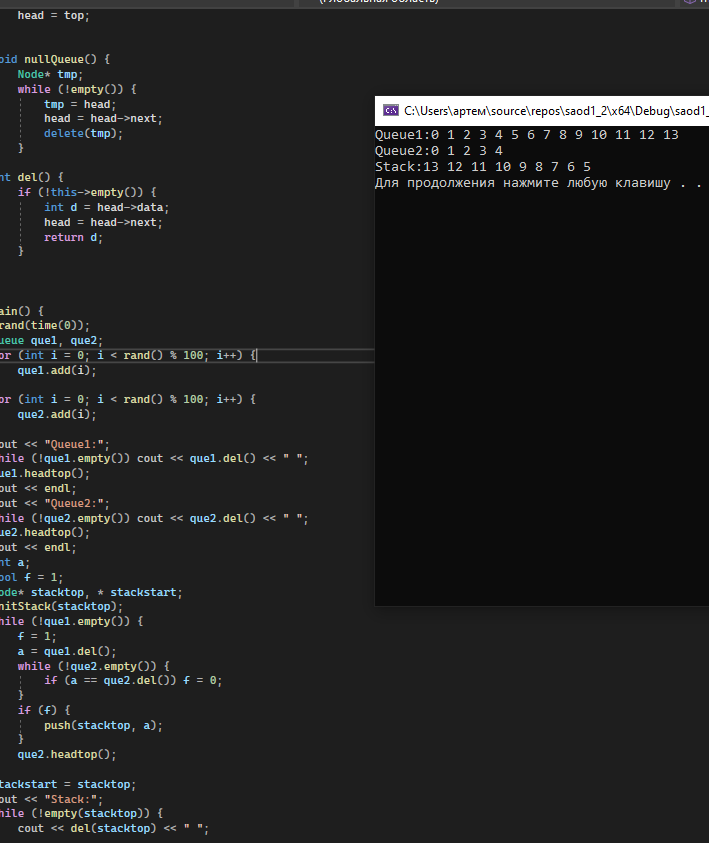
stacktop = stackstart;

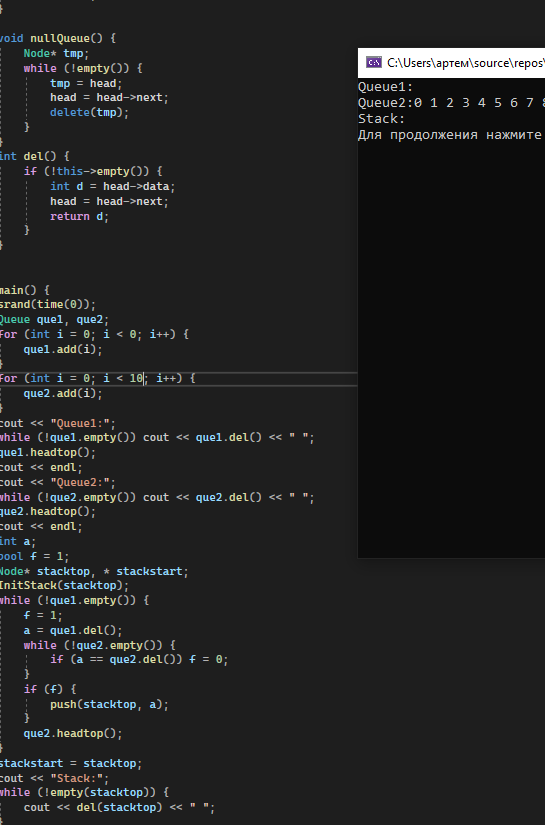
cout << endl;

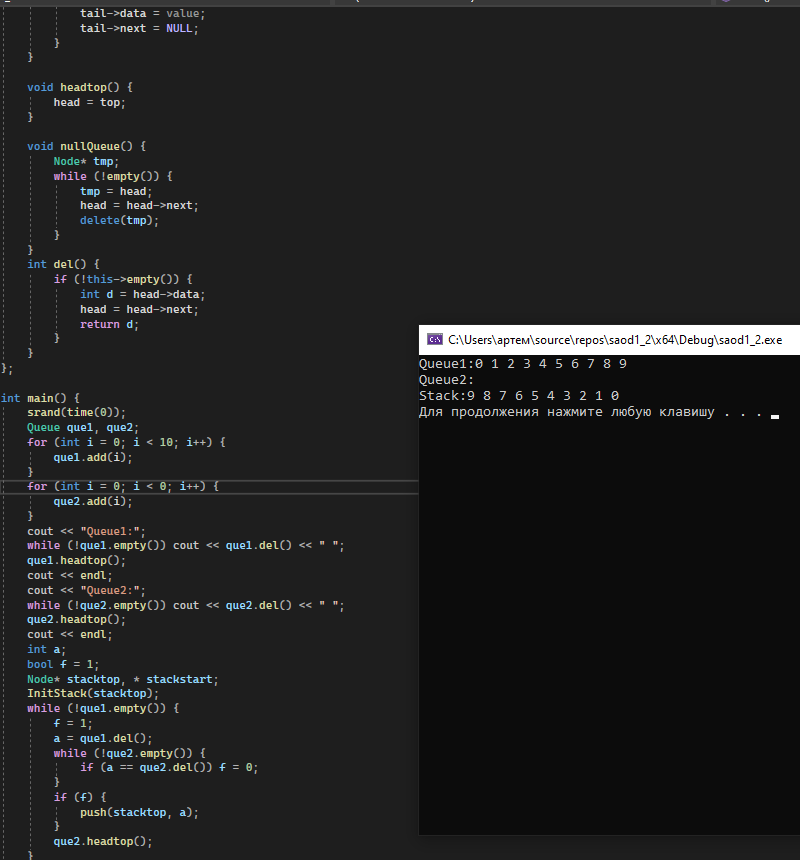
system("Pause");

}

****

****

****

****